Maximizing Network and Storage Performance for Big Data Analytics

Xiaodong Zhang

Ohio State University

Collaborators

Rubao Lee, Ying Huai, Tian Luo, Yuan Yuan Ohio State University

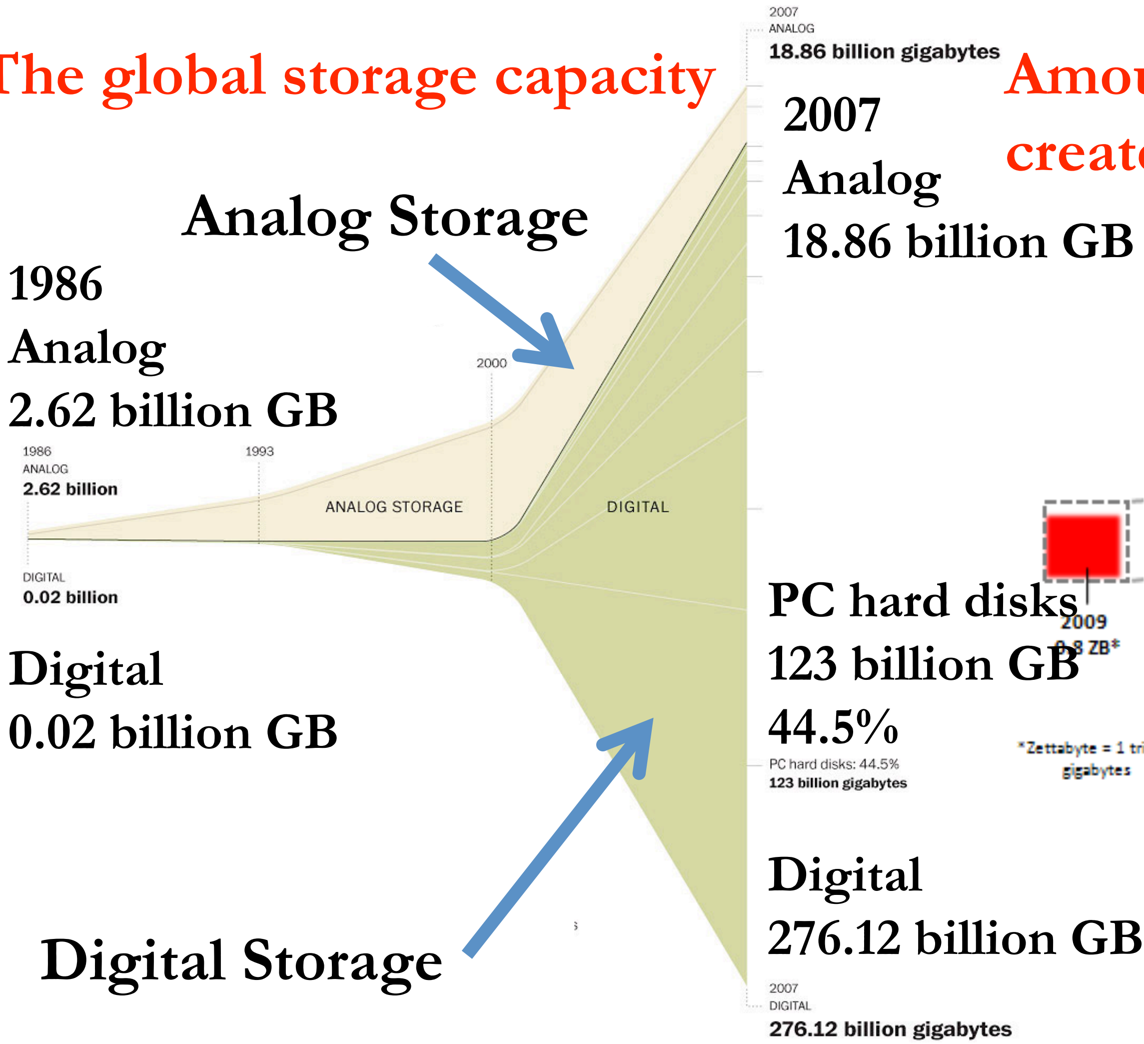
Yongqiang He and the Data Infrastructure Team, Facebook

Fusheng Wang, Emory University

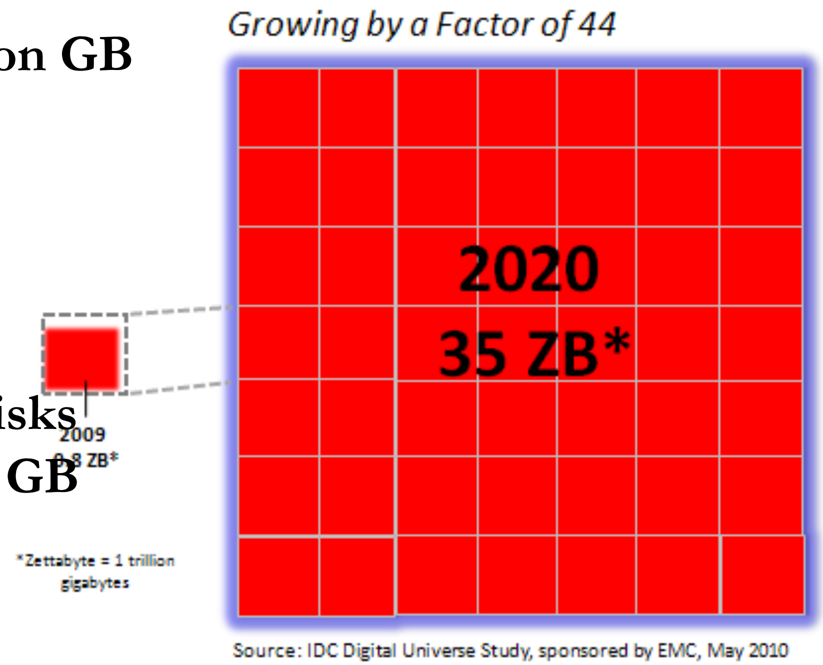
Zhiwei Xu, Institute of Comp. Tech, Chinese Academy of Sciences

Digital Data Explosion in Human Society

The global storage capacity



Amount of digital information created and replicated in a year



Source:
Exabytes: Documenting the 'digital age' and huge growth in computing capacity,
The Washington Post

Challenge of Big Data Management and Analytics (1)

- **Existing DB technology is not prepared for the huge volume**
  - Until 2007, **Facebook** had a **15TB** data warehouse by a big-DBMS-vendor
  - Now, **~70TB** compressed data added into Facebook data warehouse **every day** (4x total capacity of its data warehouse in 2007)
  - Commercial parallel DBs rarely have **100+** nodes
  - Yahoo!'s Hadoop cluster has **4000+** nodes; Facebook's data warehouse has **2750+** nodes

  - **Typical science and medical research examples:**
    - Large Hadron Collider at **CERN** generates over **15 PB** of data per year
    - Pathology Analytical Imaging Standards databases at Emory reaches **7TB,** going to PB
    - LANL Turbulence Simulation: processing the amount of data at **PB** level.

Challenge of Big Data Management and Analytics (2)

❑ Big data is about all kinds of data

- Online services (social networks, retailers …) focus on big data of online and off-line **click-stream** for deep analytics
- **Medical image** analytics are crucial to both biomedical research and clinical diagnosis

❑ Complex analytics to gain deep insights from big data

- Data mining
- Pattern recognition
- Data fusion and integration
- Time series analysis
- **Goal**: gain deep insights and new knowledge

Challenge of Big Data Management and Analytics (3-4)

- **Conventional database business model is not affordable**
  - Expensive software license
  - High maintenance fees even for open source DBs
  - Store and manage data in a system at least $10,000/TB*
  - In contrast, Hadoop-like systems only cost $1,500/TB**

- **Conventional database processing model is "scale-up" based**
  - Performance improvement relies on CPU/memory/storage/network updates in a dedicated site (**BSP model**, CACM, 1990)
  - Big data processing model is **"scale-out"** based (**DOT model**, SOCC'11): relies on continuously adding low cost computing and storage nodes in a

MapReduce programming model becomes an effective data processing engine for big data analytics

Why MapReduce?

- **A simple but effective programming model designed to process huge volumes of data concurrently**

- **Two unique properties**
  - Minimum dependency among tasks (almost **sharing nothing**)
  - Simple task operations in each node (**low cost machines** are sufficient)

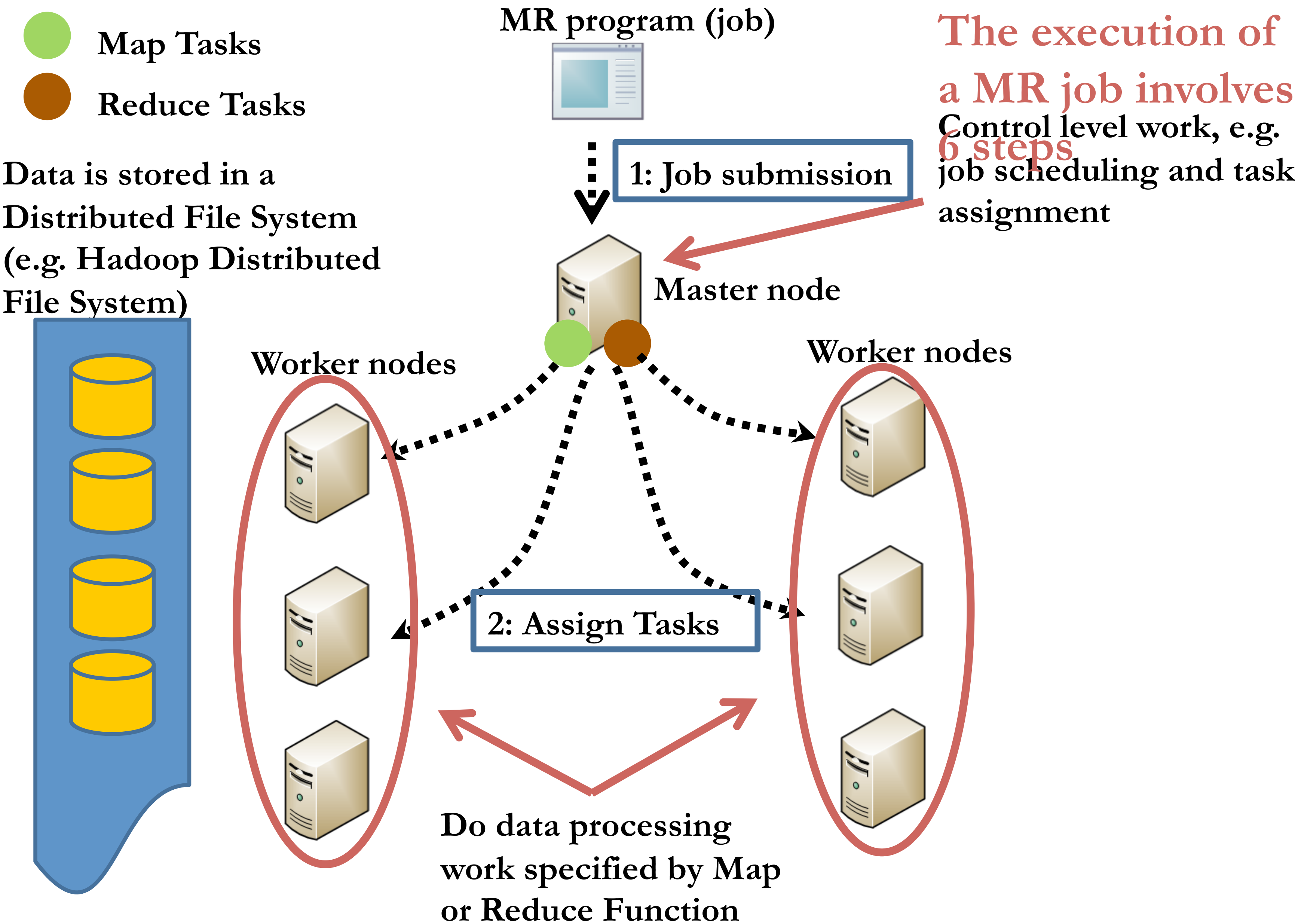
- **Two strong merits for big data anaytics**
  - **Scalability** (Amadal's Law): increase throughput by increasing # of nodes
  - **Fault-tolerance** (quick and low cost recovery of the failures of tasks)

- **Hadoop is the most widely used implementation of MapReduce**
  - in hundreds of society-dependent corporations/organizations for big data analytics: **AOL, Baidu**, **EBay**, **Facebook**, **IBM**, **NY Times**, **Yahoo!** ….

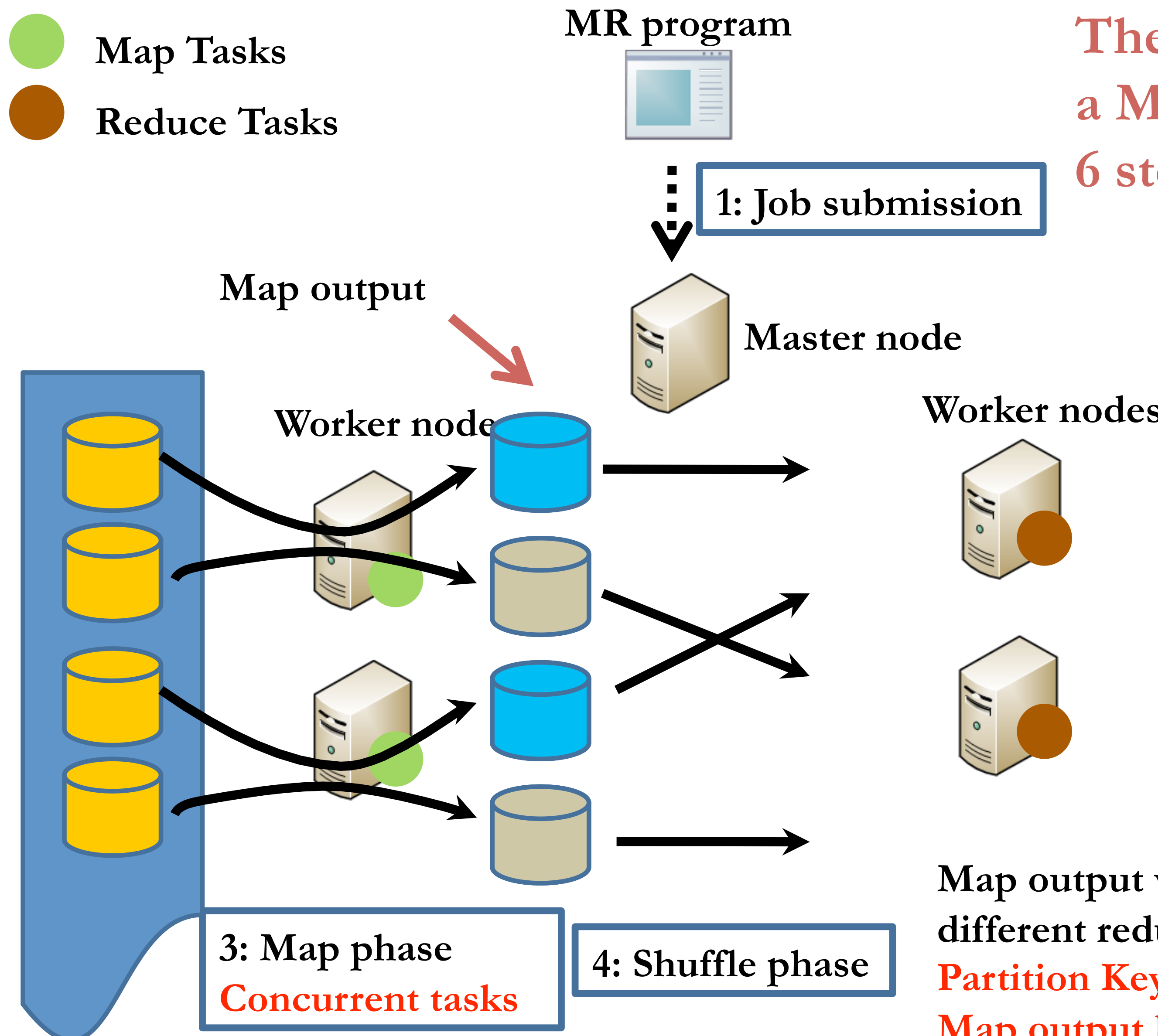
MapReduce Overview

- **The basic framework comes from functional programming: simple key/value pairs forms a chain of MR execution**
- **Map: (k1, v1) → (k2, v2)**
- **Reduce: (k2, v2) → (k3, v3)**
- **Shuffle: Partition Key (It could be the same as k2, or not)**
  - Partition Key: to determine how a key/value pair in the map output be transferred to a reduce task

MR(Hadoop) Job Execution Insights



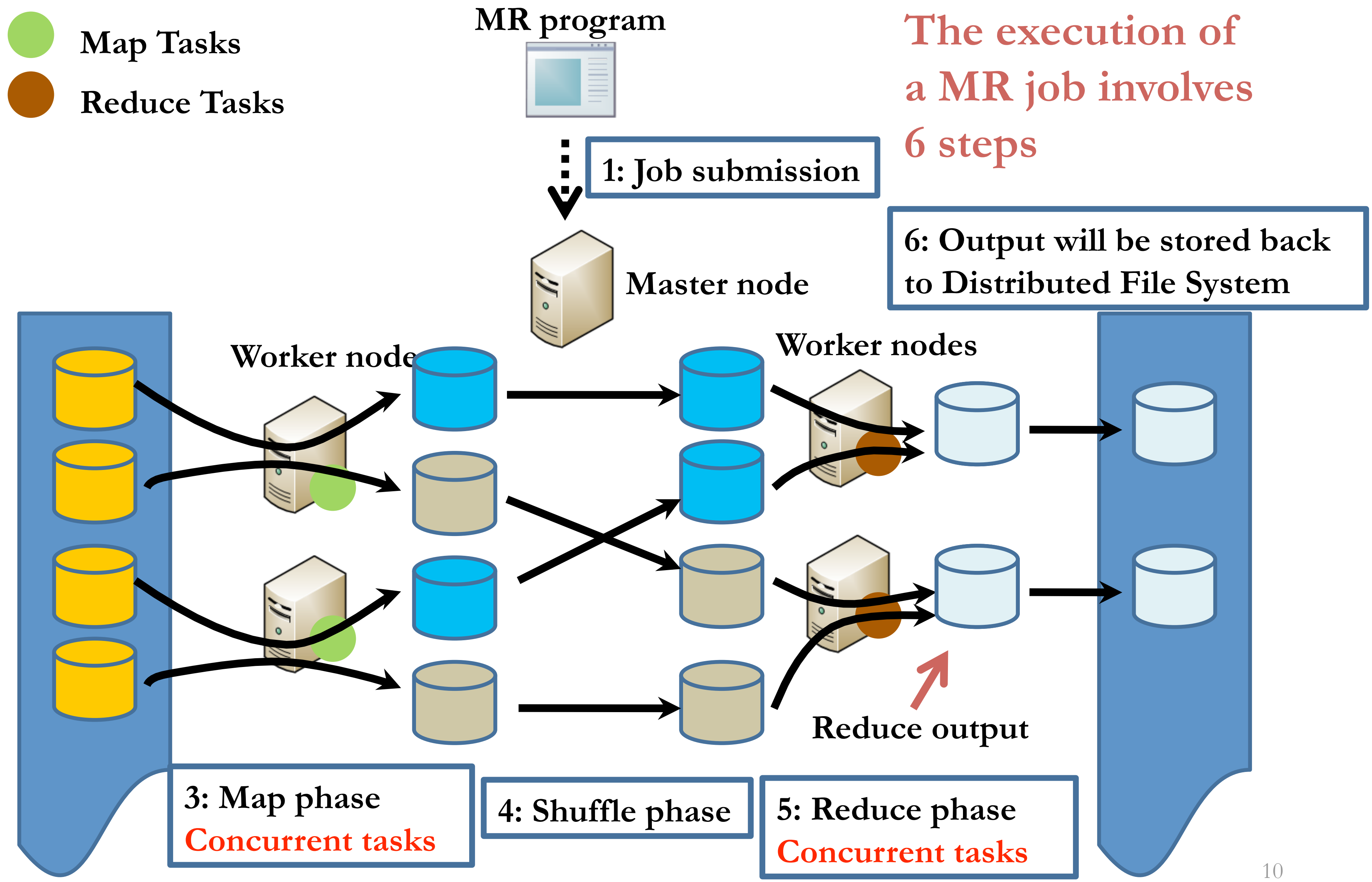
MR(Hadoop) Job Execution Insights



The execution of a MR job involves 6 steps

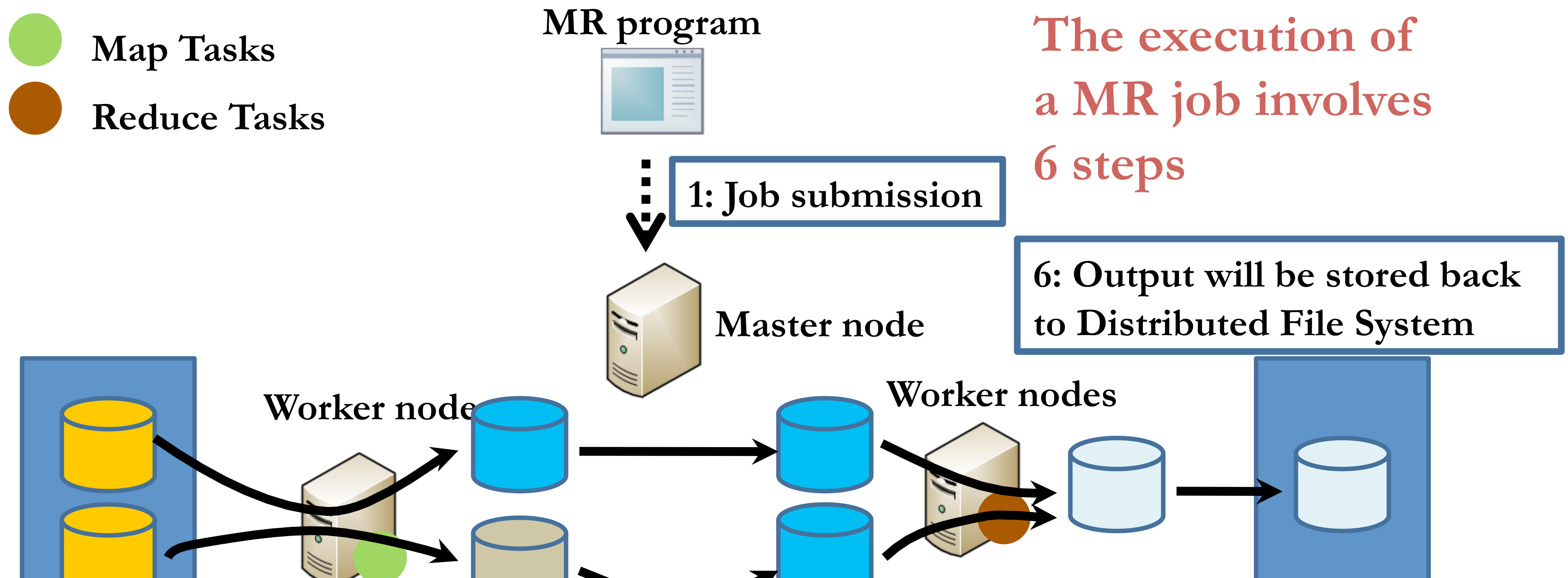
Map output will be shuffled to different reduce tasks based on Partition Keys (PKs) (usually Map output keys)

MR(Hadoop) Job Execution Insights



MR(Hadoop) Job Execution Insights

The execution of a MR job involves 6 steps



A MapReduce (MR) job is highly resource-consuming:

1: Input data scan in the Map phase => local or remote I/Os

2: Store intermediate results of Map output => local I/Os

3: Transfer data across in the Shuffle phase => network costs

4: Store final results of this MR job => local I/Os + network costs (replicate data)

Two Critical Challenges in Production Systems

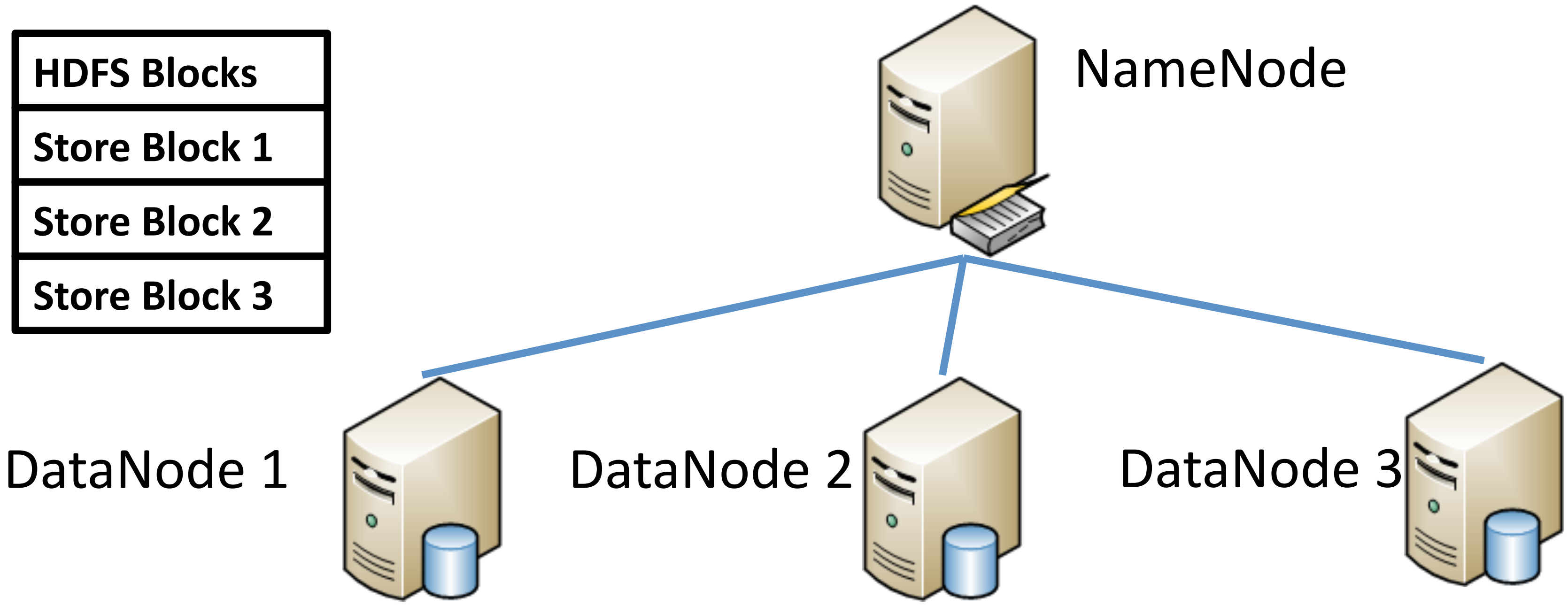
- **Background: Standard Relational Databases have been moved to MapReduce Environment, such as Hive and Pig by Facebook and Yahoo!**
- **Challenge 1: How to initially store big data in distributed systems**
  - **Objective:** to minimize network and storage costs for massive accesses
- **Challenge 2: How to automatically convert relational database queries into MapReduce jobs**
  - **Objectives:** to minimize network and storage costs for MR job execution
- **Addressing these two Challenges, we aim to achieve**
  - **High performance of big data analytics**
  - **High productivity of big data analytics**

Challenge 1: Fast and Storage-efficient Data Placement

- **Data loading (L)**
  - the overhead of writing data to distributed files ystem and local disks
- **Query processing (P)**
  - local storage bandwidths of query processing
  - the amount of network transfers
- **Storage space utilization (S)**
  - Data compression ratio
  - The convenience of applying efficient compression algorithms
- **Adaptivity to dynamic workload patterns (W)**
  - Additional overhead on certain queries

➢**Objective: to design and implement a data placement structure meeting these requirements in MapReduce-based data warehouses**

Initial Stores of Big Data in Distributed Environment



- ❑ **HDFS (Hadoop Distributed File System) blocks are distributed**
- ❑ **Users have a limited ability to specify customized data placement policy**
  - e.g. to specify which blocks should be co-located
- ❑ **Minimizing I/O costs in local disks and intra network communication**

MR programming is not that "simple"!

```
package tpch;
import java.io.IOException;
import java.util.ArrayList;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
...
import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.util.Tool;
...
        inputFile = ((FileSplit)context.getInputSplit()).
        if (inputFile.compareTo("lineitem.tbl") == 0){
            isLineitem = true;
```

```
public static class Reduce extends Reducer<IntWritable,Text,IntWritable,Text> {
        private Text result = new Text();

        public void reduce(IntWritable key, Iterable<Text> values,
                           Context context
                ) throws IOException, InterruptedException {

                    double sumQuantity = 0.0;
                    IntWritable newKey = new IntWritable();
...
                }
                else
...
                context.write(newKey, result);
            }
        }
...
        public static void main(String[] args) throws Exception {
            int res = ToolRunner.run(new Configuration(), new Q18Job1(), args);
            System.exit(res);
        }
}
```

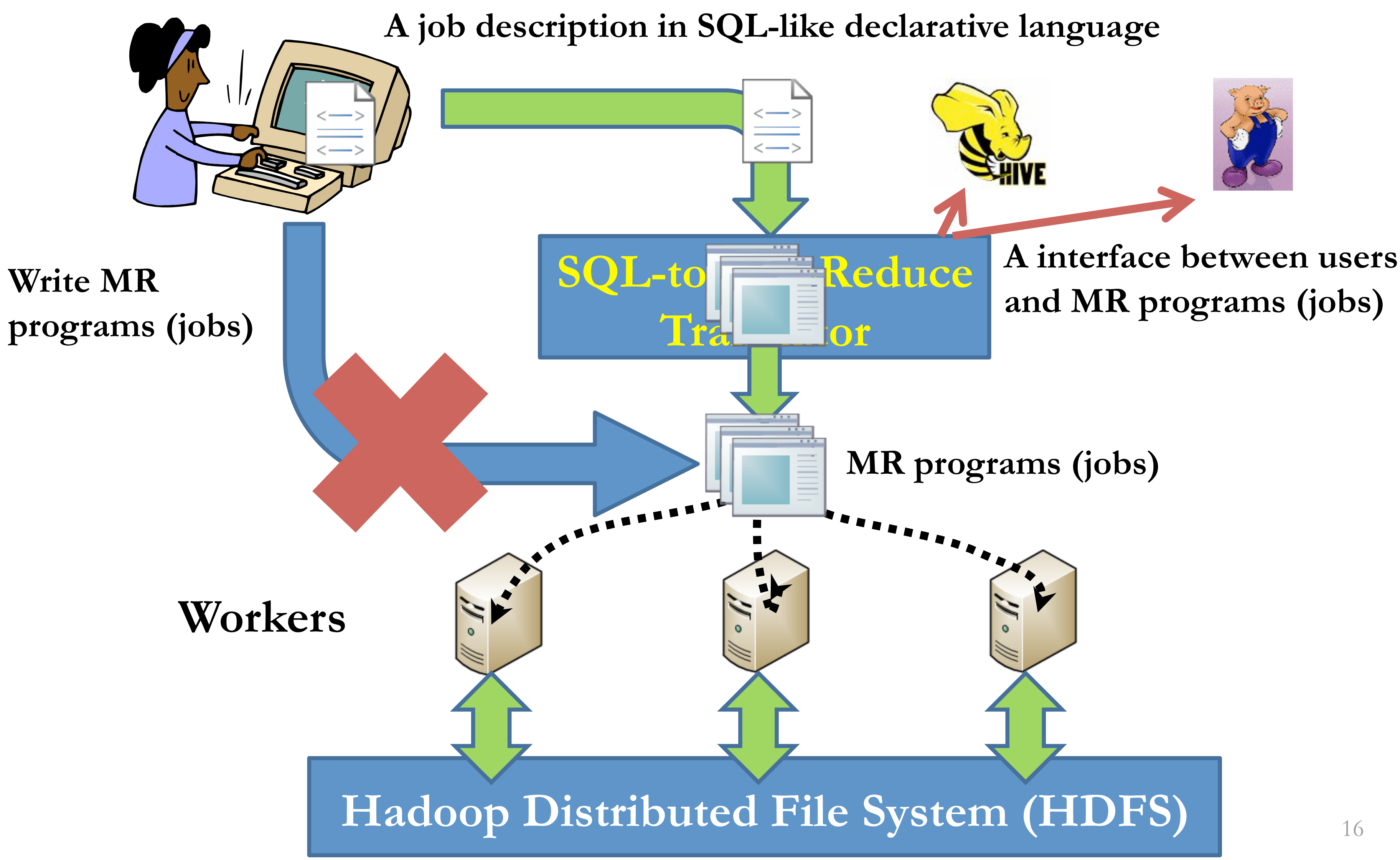
This complex code is for a simple MR job

Low Productivity!

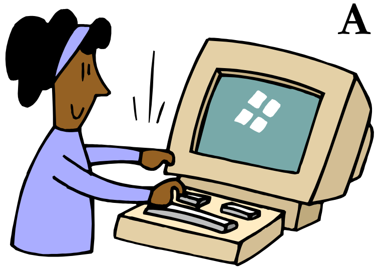
Do you miss some thing like …

"SELECT * FROM Book WHERE price > 100.00"?

Challenge 2: High Quality MapReduce in Automation



Challenge 2: High Quality MapReduce in Automation



A job description in SQL-like declarative language

Write MR programs (jobs)

SQL-to-MapReduce Translator

A interface between users and MR programs (jobs)

A data warehousing system **(Facebook)**

A high-level programming environment **(Yahoo!)**

Improve productivity from hand-coding MapReduce programs

- **95%+** Hadoop jobs in Facebook are generated by **Hive**
- **75%+** Hadoop jobs in Yahoo! are invoked by **Pig***

* http://hadooplondon.eventbrite.com/

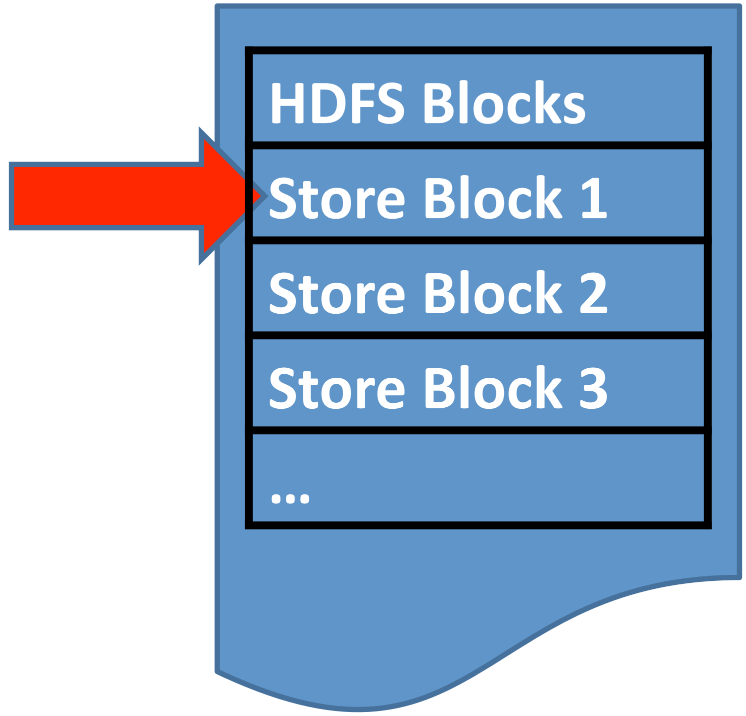
Outline

- **RCFile: a fast and space-efficient placement structure**
  - Re-examination of existing structures
  - A Mathematical model as basis of RCFile
  - Experiment results
- **Ysmart: a high efficient query-to-MapReduce translator**
  - Correlations-aware is the key
  - Fundamental Rules in the translation process
  - Experiment results
- **Impact of RCFile and Ysmart in production systems**
- **Conclusion**

Row-Store: Merits/Limits with MapReduce

Table

A	B	C	D
101	201	301	401
102	202	302	402
103	203	303	403
104	204	304	404
105	205	305	405

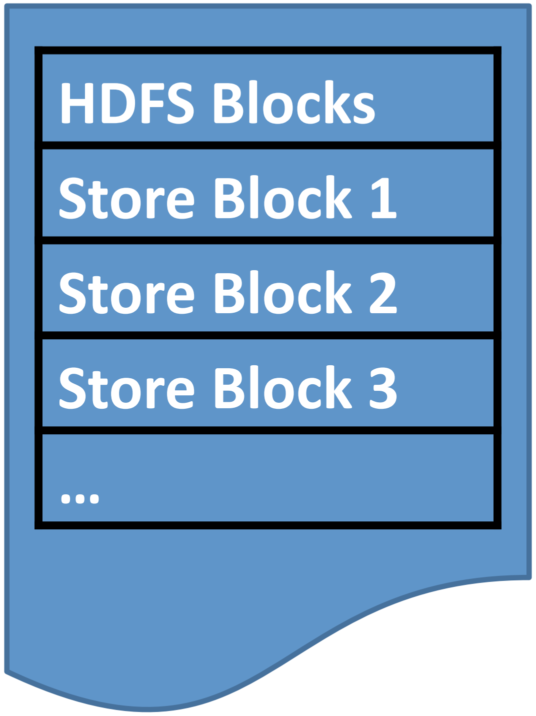


- Data loading is fast (no additional processing);
- All columns of a data row are located in the same HDFS block
- Not all columns are used (unnecessary storage bandwidth)
- Compression of different types may add additional overhead

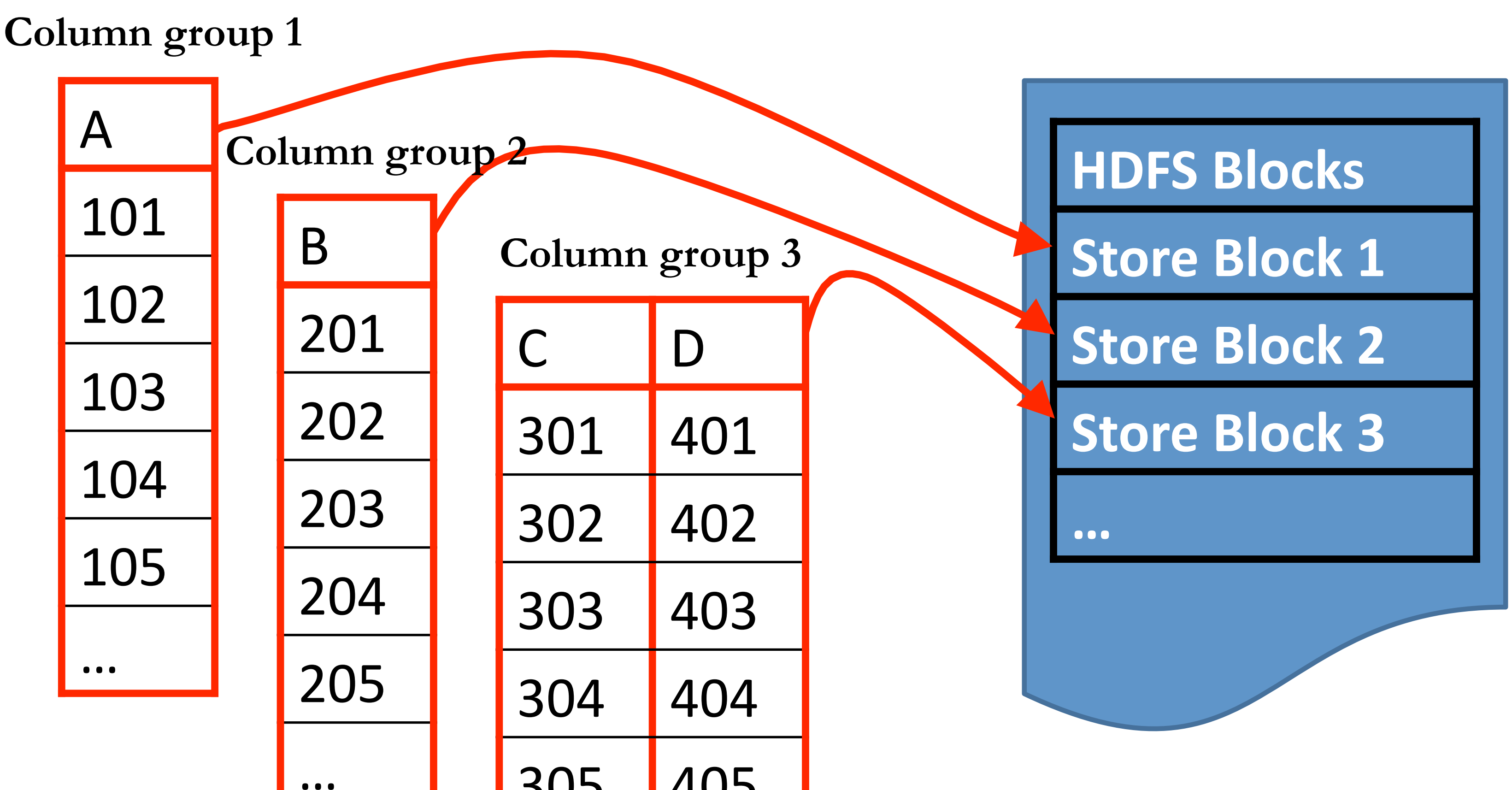
Column-Store: Merits/Limits with MapReduce

Table

A	B	C	D
101	201	301	401
102	202	302	402
103	203	303	403
104	204	304	404
105	205	305	405
…	…	…	…



Column-Store: Merits/Limits with MapReduce



- ➢Unnecessary I/O costs can be avoided: Only needed columns are loaded, and easy compression
- ➢Additional network transfers for column grouping

Optimization of Data Placement Structure

- **Consider four processing requirements comprehensively**
- **The optimization problem in systems design becomes:**
  - In a environment of **dynamic workload (W)** and with a **suitable data compression algorithm (S)** to improve the utilization of data storage,

    find a data placement structure **(DPS)** that minimizes the processing time of a basic operation **(OP)** on a table **(T)** with ***n*** columns
- **Two basic operations**
  - **Write:** the essential operation of **data loading (L)**
  - **Read:** the essential operation of **query processing (P)**

Finding Optimal Data Placement Structure

$$E(read \mid DPS) =$$

$$\sum_{i=1}^{n} \sum_{j=1}^{\binom{n}{i}} f(i,j,n)\left(\frac{S}{B_{local}} \times \frac{1}{\rho} \times \alpha(DPS) + \lambda(DPS,i,j,n) \times \frac{S}{B_{netwotk}} \times \frac{i}{n}\right)$$

	Row-Store	Column-store	**Ideal**
Read efficiency	**1**	i/n (*optimal*)	i/n (*optimal*)
Communication overhead	**0** (*optimal*)	β $(0\% \leq \beta \leq 100\%)$	**0** (*optimal*)

Can we find a Data Placement Structure with both optimal *read efficiency* and *communication overhead* ?

Goals of RCFile

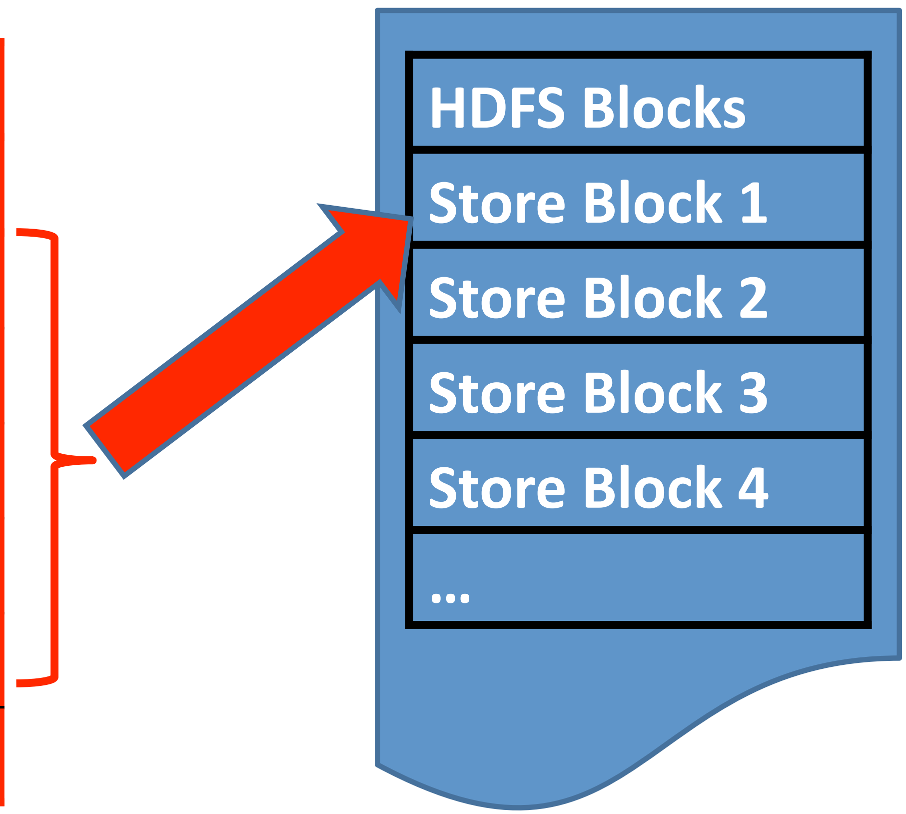
- ❑ **Eliminate unnecessary I/O costs like Column-store**
  - Only read needed columns from disks
- ❑ **Eliminate network costs in row construction like Row-store**
- ❑ **Keep the fast data loading speed of Row-store**
- ❑ **Can apply efficient data compression algorithms conveniently like Column-store**
- ❑ **Eliminate all the limits of Row-store and Column-store**

RCFile: Partitioning a Table into Row Groups

A HDFS block consists of one or multiple row groups

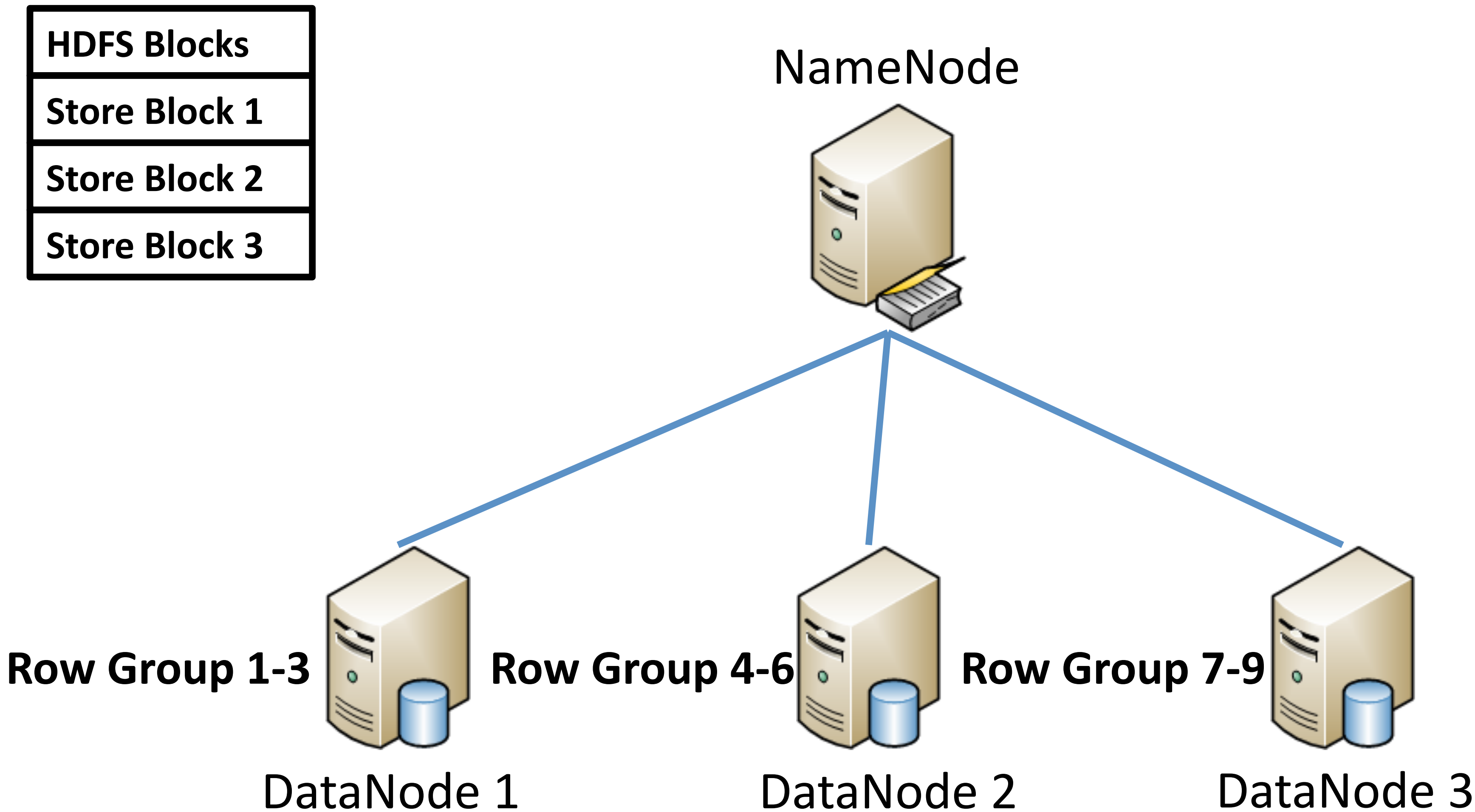
Table

A	B	C	D
…	A.Row	Group	…
101	201	301	401
102	202	302	402
103	203	303	403
104	204	304	404
105	205	305	405
…	…	…	…



RCFile: Distributed Row-Group Data among Nodes

For example, each HDFS block has three row groups



Inside a Row Group

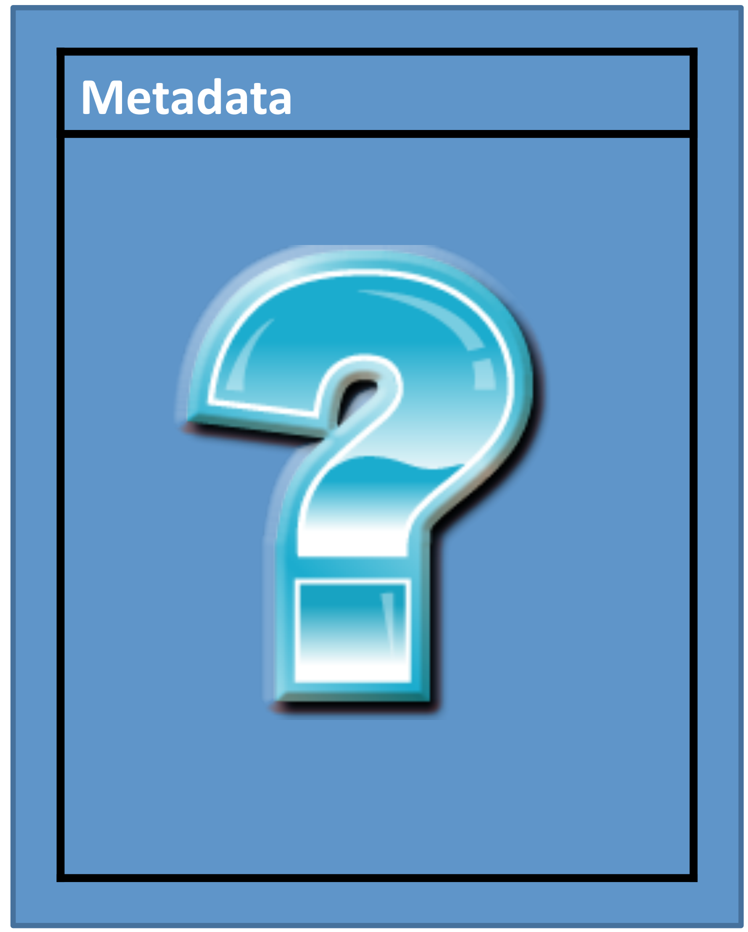
Store Block 1

101	201	301	401
102	202	302	402
103	203	303	403
104	204	304	404
105	205	305	405

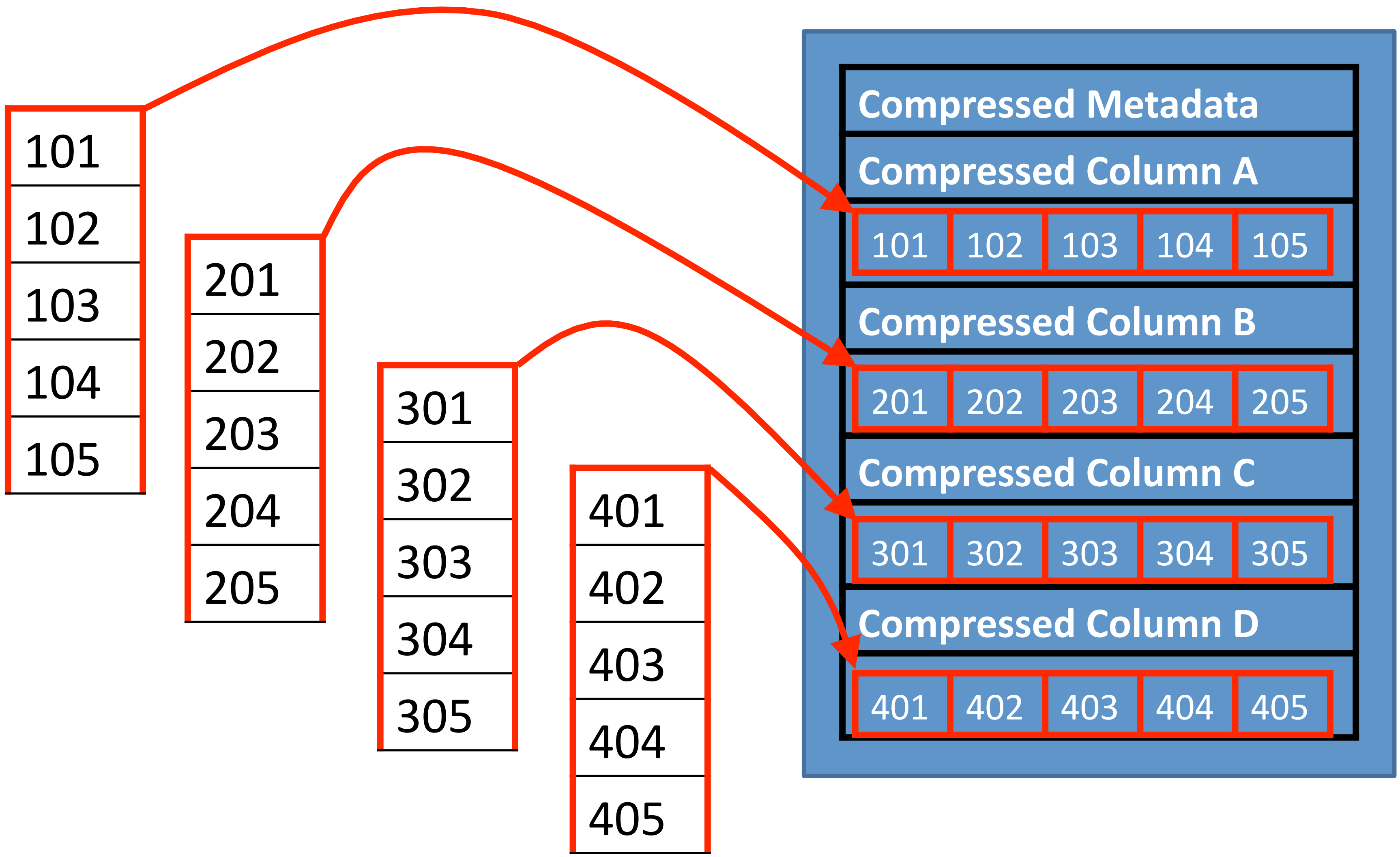
Metadata

Inside a Row Group

101	201	301	401
102	202	302	402
103	203	303	403
104	204	304	404
105	205	305	405



RCFile: Inside each Row Group



Benefits of RCFile

- **Eliminate unnecessary I/O costs**
  - In a row group, table is partitioned by columns
  - Only read needed columns from disks
- **Eliminate network costs in row construction**
  - All columns of a row are located in the same HDFS block
- **Comparable data loading speed to Row-Store**
  - Only adding a vertical-partitioning operation in the data loading procedure of Row-Store
- **Can apply efficient data compression algorithms conveniently**
  - Can use compression schemes used in Column-store

Expected Time of a Read Operation

$$E(read \mid DPS) =$$

$$\sum_{i=1}^{n} \sum_{j=1}^{\binom{n}{i}} f(i,j,n)\left(\frac{S}{B_{local}} \times \frac{1}{\rho} \times \alpha(DPS) + \lambda(DPS,i,j,n) \times \frac{S}{B_{netwotk}} \times \frac{i}{n}\right)$$

	Row-Store	Column-store
Read efficiency	**1**	i/n (*optimal*)
Communication overhead	**0** (*optimal*)	β ($0\% \leq \beta \leq 100\%$)

Expected Time of a Read Operation

$$E(read \mid DPS) =$$

$$\sum_{i=1}^{n} \sum_{j=1}^{\binom{n}{i}} f(i, j, n)\left(\frac{S}{B_{local}} \times \frac{1}{\rho} \times \alpha(DPS) + \lambda(DPS, i, j, n) \times \frac{S}{B_{netwotk}} \times \frac{i}{n}\right)$$

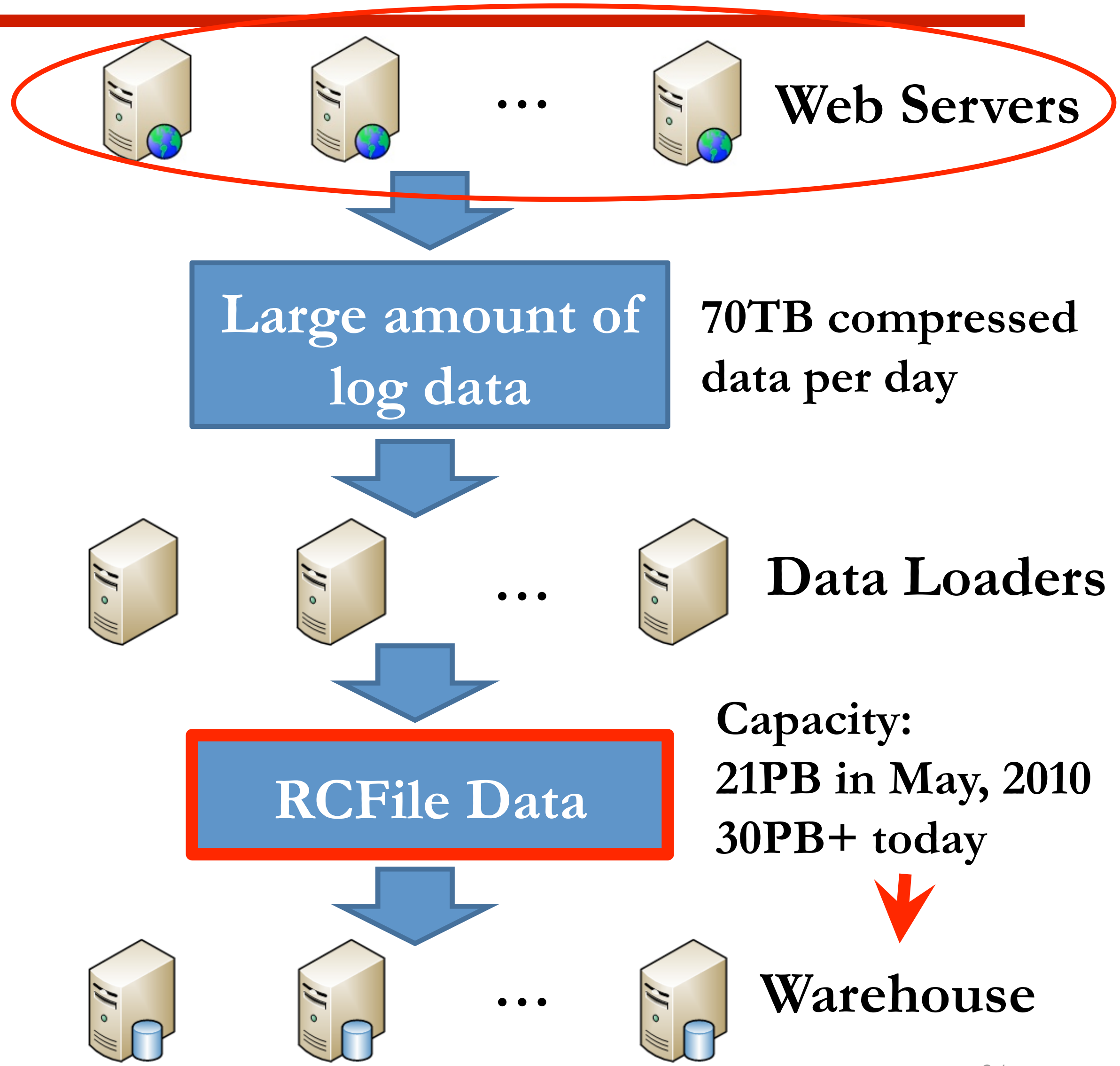
	Row-Store	Column-store	RCFile
Read efficiency	**1**	i/n (*optimal*)	i/n (optimal)
Communication overhead	**0** (*optimal*)	β ($0\% \le \beta \le 100\%$)	**0** (optimal)

Facebook Data Analytics Workloads Managed By RCFile

- **Reporting**
  - E.g. daily/weekly aggregations of impression/click counts
- **Ad hoc analysis**
  - E.g. geographical distributions and activities of users in the world
- **Machine learning**
  - E.g. online advertizing optimization and effectiveness studies
- **Many other data analysis tasks on user behavior and patterns**
- **User workloads and related analysis cannot be published**
- **RCFile evaluation with public available workloads with excellent performance (ICDE'11)**

RCFile in Facebook

The interface to
500+ million users



Picture source: Visualizing Friendships, http://www.facebook.com/notes/facebook-engineering/visualizing-friendships/469716398919

Summary of RCFile

- Data placement structure lays a foundation for MapReduce-based big data analytics
- Our optimization model shows RCFile meets all basic requirements
- RCFile: an operational system for daily tasks of big data analytics
  - A part of **Hive**, a data warehouse infrastructure on top of Hadoop.
  - A default option for **Facebook** data warehouse
  - Has been integrated into **Apache Pig** since version 0.7.0 (expressing data analytics tasks and producing MapReduce programs)
  - **Customized RCFile** systems for special applications
- Refining RCFile and optimization model, making RCFile as a standard data placement structure for big data analytics

Outline

- ❑ ~~RCFile: a fast and space-efficient placement structure~~
  - ~~Re-examination of existing structures~~
  - ~~A Mathematical model as basis of RCFile~~
  - ~~Experiment results~~
- ❑ **Ysmart: a high efficient query-to-MapReduce translator**
  - Correlations-aware is the key
  - Fundamental Rules in the translation process
  - Experiment results
- ❑ **Impact of RCFile and Ysmart in production systems**
- ❑ **Conclusion**

Translating SQL-like Queries to MapReduce Jobs: Existing Approach

- **"Sentence by sentence" translation**
  - [C. Olston et al. SIGMOD 2008], [A. Gates et al., VLDB 2009] and [A. Thusoo et al., ICDE2010]
  - Implementation: Hive and Pig

- **Three steps**
  - **Identify major sentences with operations that shuffle the data**
    - Such as: Join, Group by and Order by
  - **For every operation in the major sentence that shuffles the data, a corresponding MR job is generated**
    - e.g. a join op. => a join MR job
  - **Add other operations, such as selection and projection, into corresponding MR jobs**

Existing SQL-to-MapReduce translators give unacceptable performance.

An Example: TPC-H Q21

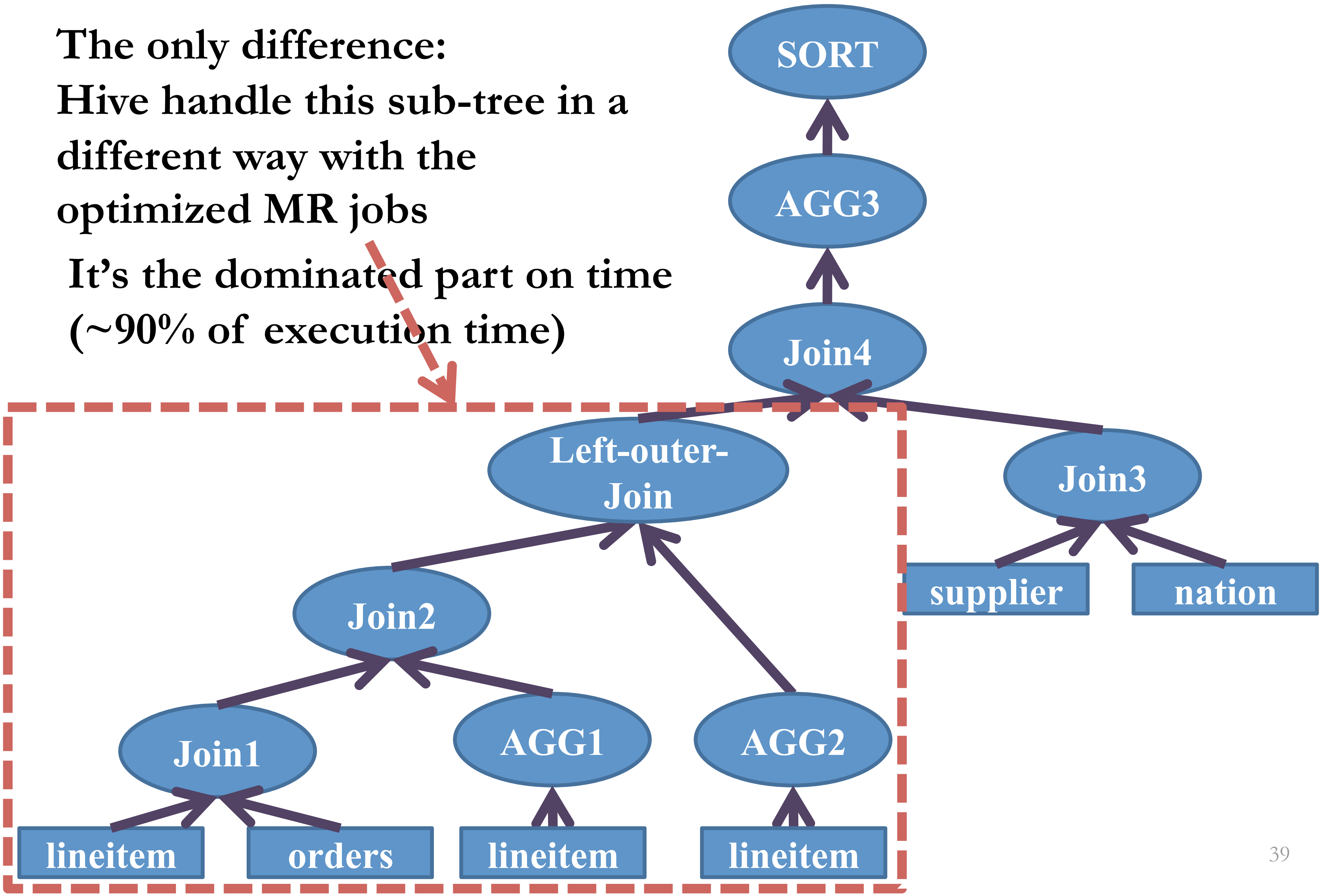
- One of the most complex and time-consuming queries in the TPC-H benchmark for data warehousing performance
- Optimized MR Jobs vs. Hive in a Facebook production cluster

What's wrong?

The Execution Plan of TPC-H Q21

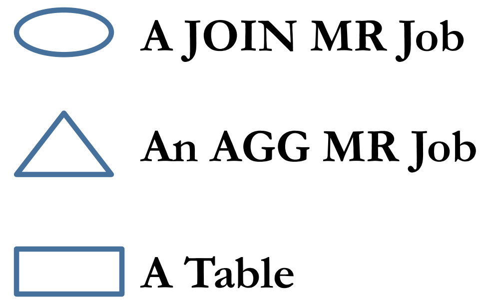
The only difference:
Hive handle this sub-tree in a different way with the optimized MR jobs

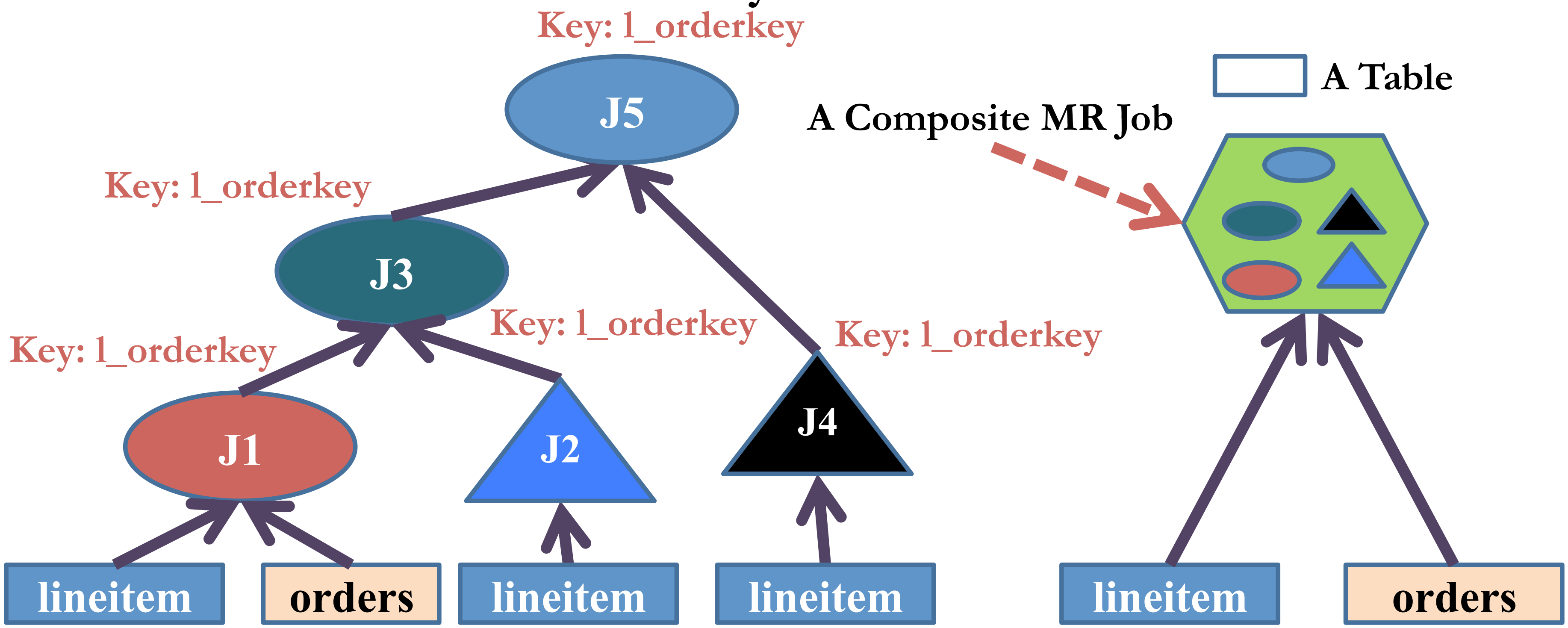
It's the dominated part on time (~90% of execution time)



However, inter-job correlations exist.

Let's look at the Partition Key





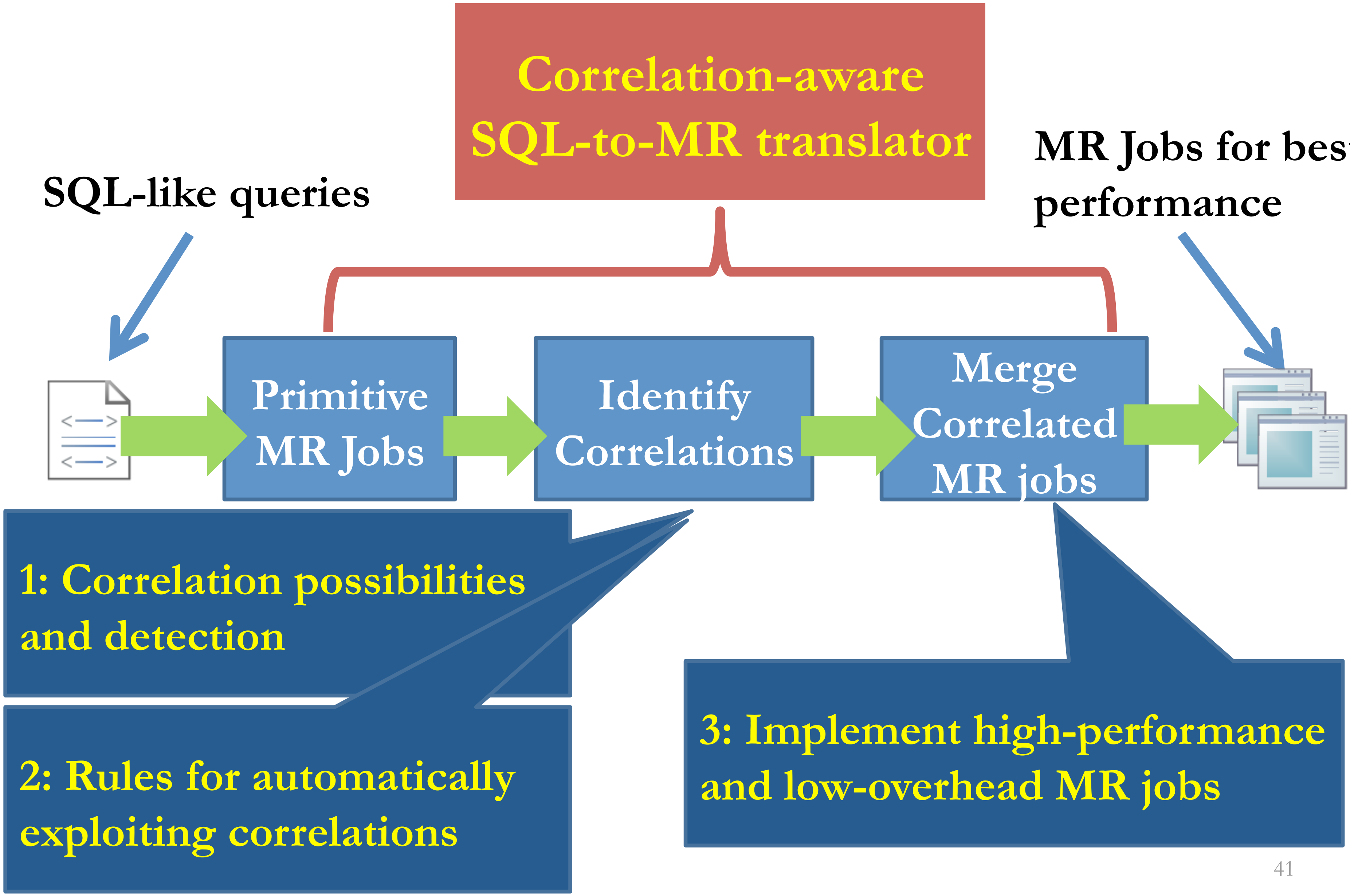
J1 to J5 all use the same partition key 'l_orderkey'

What's wrong with existing SQL-to-MR translators?

Existing translators are correlation-unaware

1. Ignore common data input
2. Ignore common data transition

Our Approaches and Critical Challenges



Query Optimization Rules for Automatically Exploiting Correlations

- Exploiting both Input Correlation and Transit Correlation

- Exploiting the Job Flow Correlation associated with Aggregation jobs

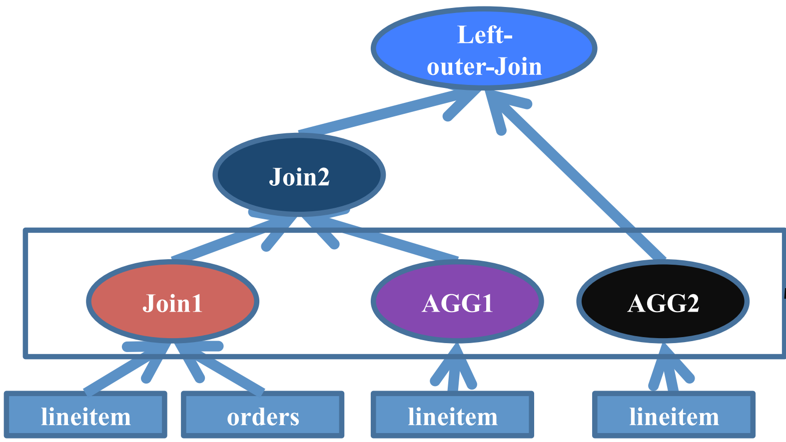
- Exploiting the Job Flow Correlation associated with JOIN jobs and their Transit Correlated parents jobs

- Exploiting the Job Flow Correlation associated with JOIN jobs

Exp1: Four Cases of TPC-H Q21

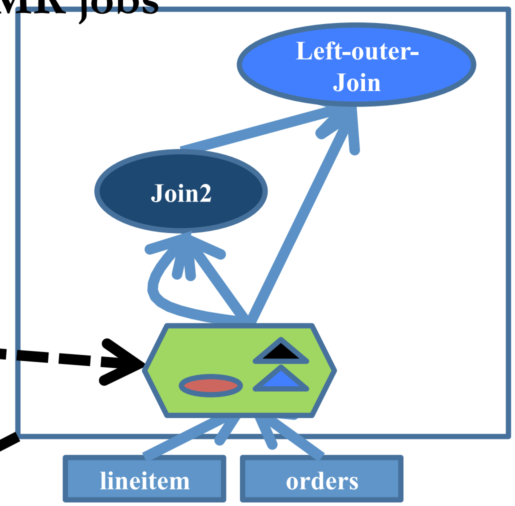
1: Sentence-to-Sentence Translation

- 5 MR jobs



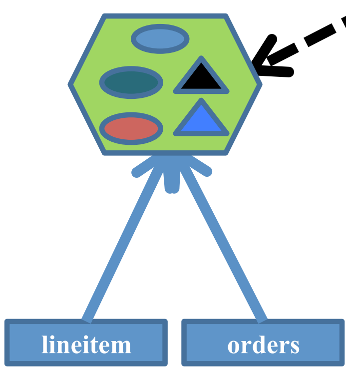
2: InputCorrelation+TransitCorrelation

- 3 MR jobs



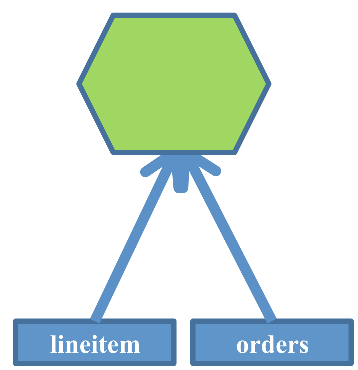
3: InputCorrelation+TransitCorrelation+ JobFlowCorrelation

- 1 MR job

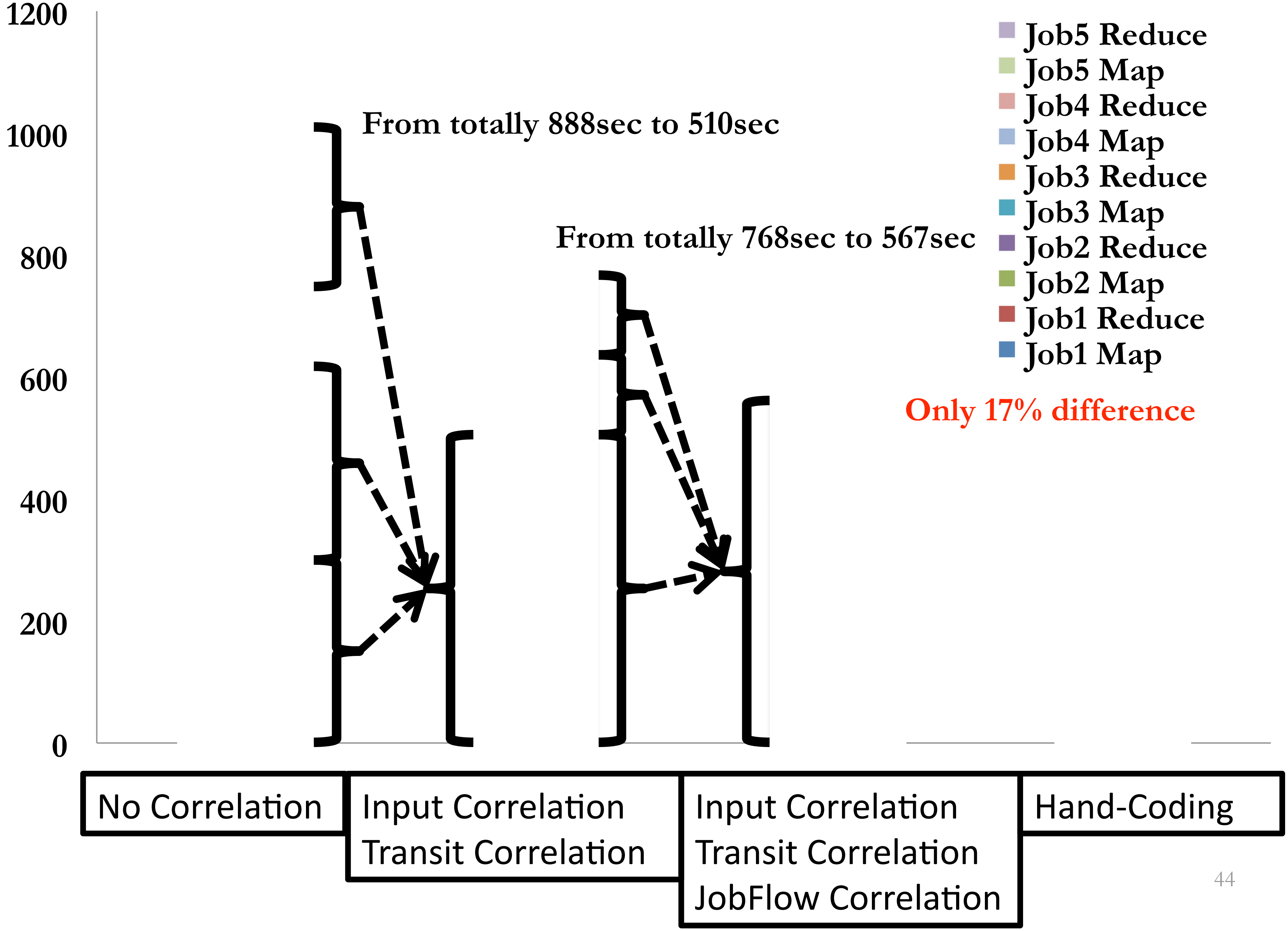


4: Hand-coding (similar with Case 3)

- In reduce function, we optimize code according query semantic

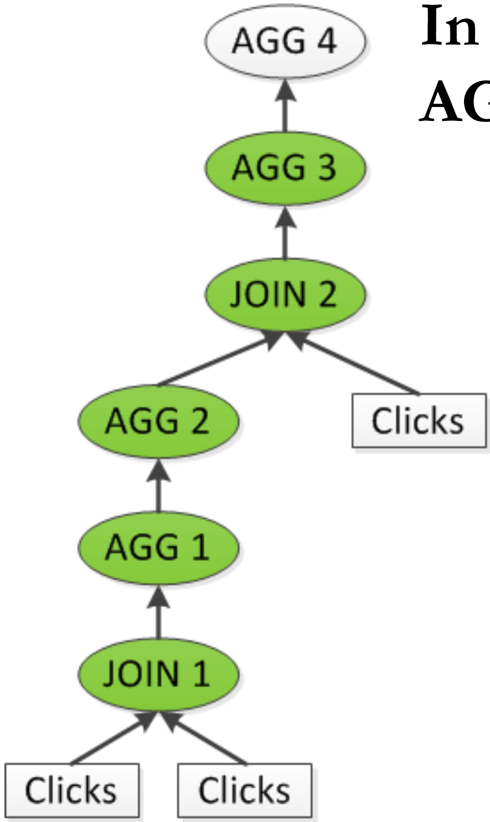


Breakdowns of Execution Time (sec)

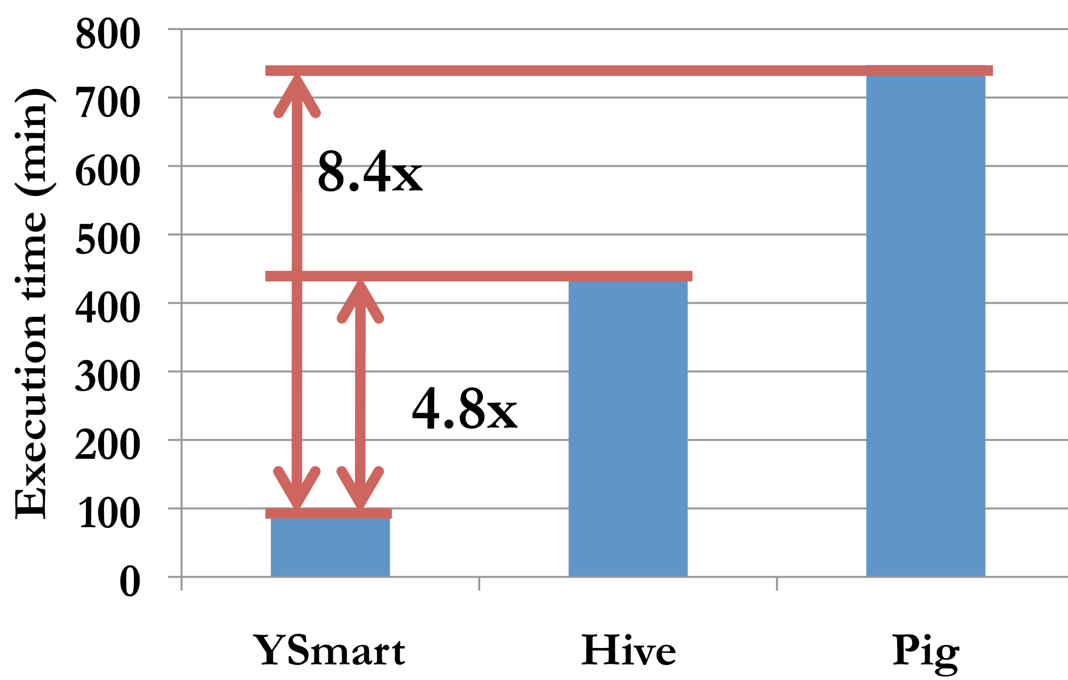


Exp2: Clickstream Analysis

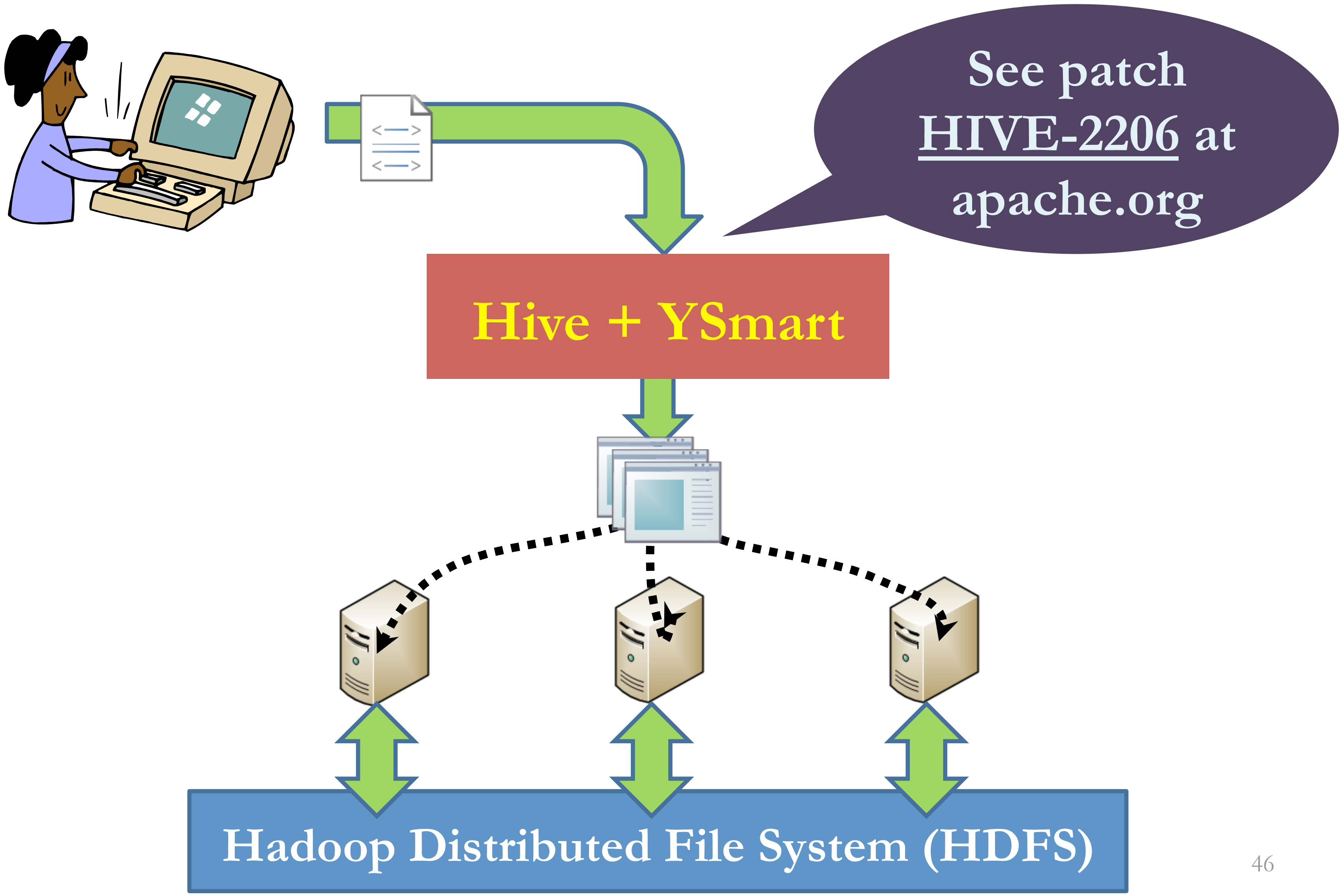
A typical query in production clickstream analysis: *"what is the average number of pages a user visits between a page in category '**X**' and a page in category '**Y**'?"*



In YSmart JOIN1, AGG1, AGG2, JOIN2 and AGG3 are executed in a single MR job

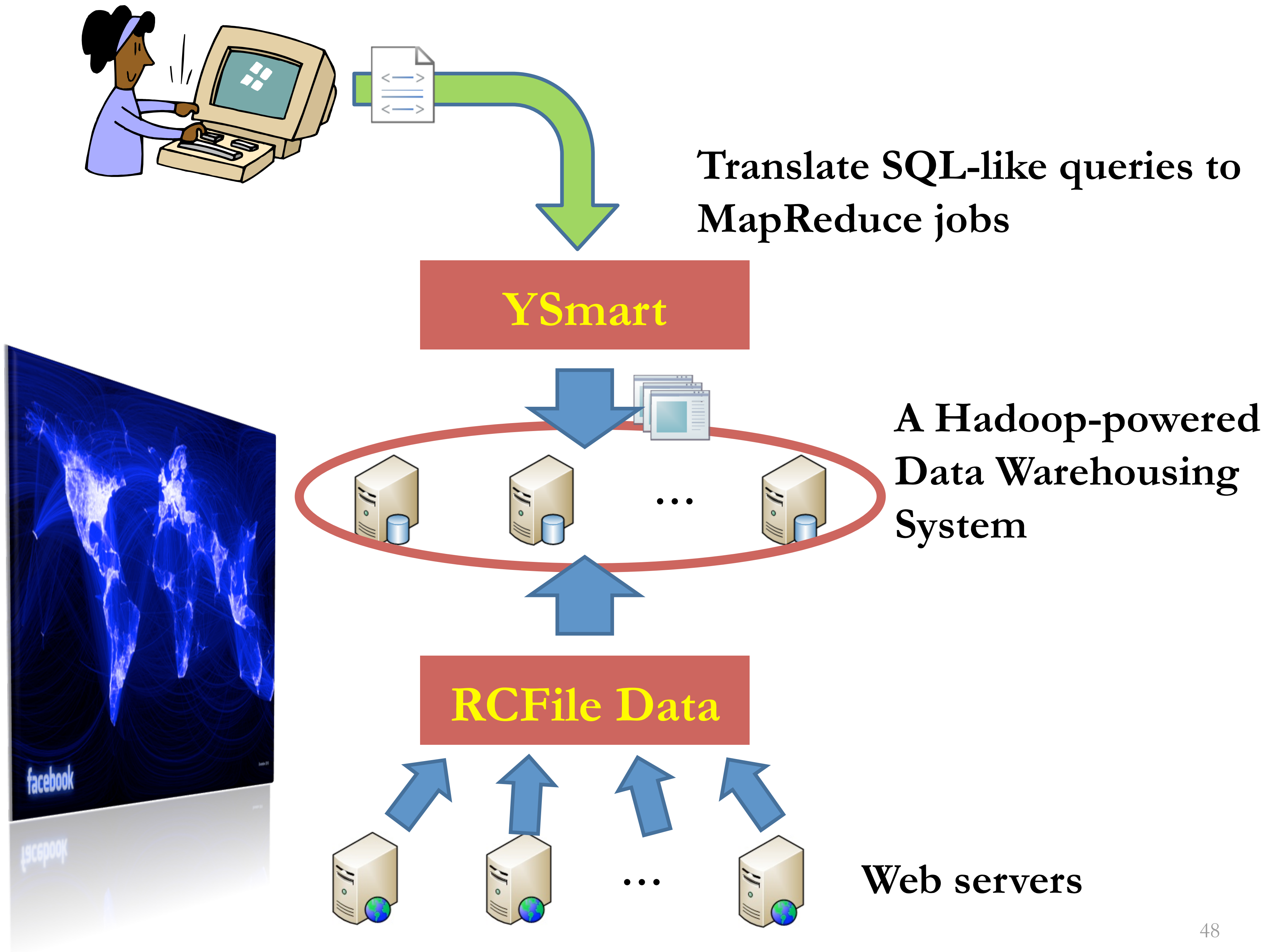


YSmart in the Hadoop Ecosystem



Summary of YSmary

- YSmart is a correlation-aware SQL-to-MapReduce translator
- Ysmart can outperform Hive by 4.8x, and Pig by 8.4x
- YSmart is being integrated into Hive
- The individual version of YSmart will be released soon


Translate SQL-like queries to MapReduce jobs
YSmart
A Hadoop-powered Data Warehousing System
…
RCFile Data
…
Web servers
facebook

Conclusion

- We have contributed two important system components: RCFile and Ysmart in the critical path of Big Data analytics Ecosystem.
- The ecosystem of Hadoop-based big data analytics is created: Hive and Pig, will soon merge into an unified system
- RCFile and Ysmart are in the critical path in such a new Ecosystem.

Thank You!